CS112 –
Java
Programming

Fall 2022

Object Oriented Programming

Notices…

Subscription to zyBooks is available. OPTIONAL online text with lots of practice exercises.
How to get access:

1. Create an account at learn.zybooks.com if you do not have one yet
2. Sign in
3. Enter zyBook code: USFCACS112HaskellFall2022
4. Subscribe. Cost is $77 for the semester. You lose access after Jan 15, 2023.

Do run "javac" and "java" separately. Running "java MyProgram.java" does not always give correct results.

- **javac MyProgram.java**
- **java MyProgram** (NOT MyProgram.java)

So far…

We have learned about built-in data types and variables

We have learned some basic statements, including assignments and mathematical operations. And println() of course…

We saw the program control statements while() and if()-else

These are the core of Procedural Programming

Why isn't this enough?

Problem Solving

The purpose of writing a program is to solve a problem

Solving a problem consists of multiple activities:

- Understand the problem
- Design a solution
- Consider alternatives and refine the solution
- Implement the solution
- Test the solution

These activities are not purely linear – they overlap and interact

Problem Solving

The key to designing a solution is breaking it down into manageable pieces

When writing software, we design separate pieces that are responsible for certain parts of the solution

Procedural Programming

Procedural Programming is not enough

- Not simple enough
- Not robust enough

…in cases where there can be hundreds or more types of user inputs (e.g. a GUI)

…in cases where dozens (or thousands) of people develop SW for the same project

…in cases where we must simplify the complexity of a sophisticated problem

Object Oriented Programming

Two key concepts

- **Class** – a user-defined data type that includes both data (variables, properties) and functions (methods, procedures)
- **Objects** – user-created variables whose type is some class. An object represents some real, important entity in our solution to some problem.

What's the big(gest) deal?

- **Creators** of a class can protect data from accidental corruption and can hide complexity…
- …meaning **Users** of a class can use the class confidently, without fear of breaking something by accident
- …and **Users** can think about their problem at a higher level. E.g. Math.log()

Encapsulation

WE WILL TALK MORE ABOUT THIS

Examples

We define a **class** to represent a student and **objects** represent actual people

- We create methods to add and drop courses, calculate GPA, print current schedule, add course grades, save changes to a central database
- We require an instructor password to add grades, and administrator password to add/drop courses, student password to see GPA

A **class** represents an image and **objects** are actual images

- We create methods to save an image as a JPEG file, increase/decrease brightness, resize the image, etc
- We do not allow the brightness to become negative, the size to become negative, etc

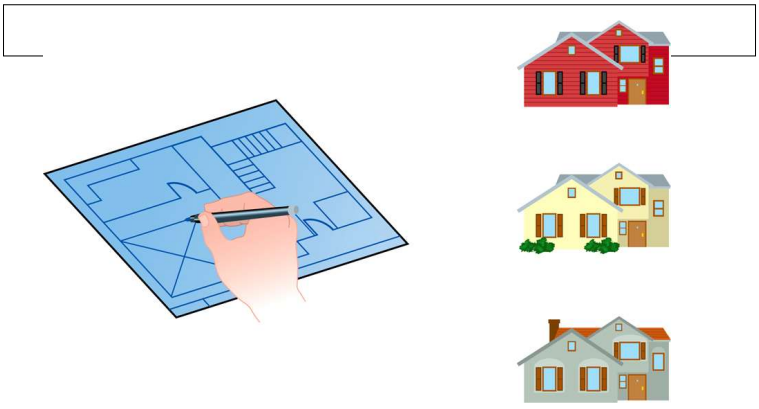
A **class** represents an item owned by the campus library. An **object** represents a particular item

- We store the item's title, publisher, publication date, length, etc
- We store whether the item is checked out and if so by whom and when due
- We store whether the item is physical (paper) or electronic
- We store its (physical or virtual) location

What programs might use this class? Course registration, signing up for dorm rooms, selling basketball tickets, figuring out who is graduating, how big classrooms are needed for classes, all sorts of things. These programs USE (but do not WRITE) class Student

Class = Blueprint

One blueprint to create several similar, but different, houses:



Classes vs Object

Data Types vs Variables – same thing

class Airline → Southwest, Delta, United, Virgin, American, KLM;

short → ageOfPaulsBike, numberOfFlatTires, cupsOfCoffeeToday;

```
class FirstProgram {
      static public void main(String[] args) {
            System.out.println("Hello world");
      }
}
// Defined a class, but no member variables and only one method
```

Examples of Classes

Class	Attributes	Operations
Student	Name Address Major Grade point average	Set address Set major Compute grade point average
Rectangle	Length Width Color	Set length Set width Set color
Aquarium	Material Length Width Height	Set material Set length Set width Set height Compute volume Compute filled weight
Flight	Airline Flight number Origin city Destination city Current status	Set airline Set flight number Determine status
Employee	Name Department Title Salary	Set department Set title Set salary Compute wages Compute bonus Compute taxes

Writing Classes

The programs we've written in previous examples have used classes defined in the Java standard class library

Now we will begin to design programs that rely on classes that we write ourselves

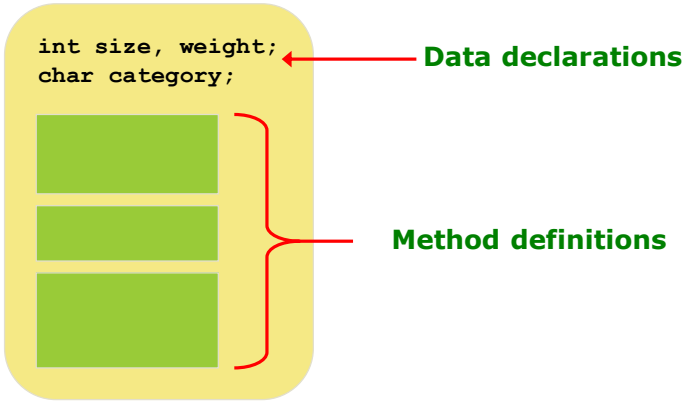
The class that contains the `main` method is just the starting point of a program

True object-oriented programming is based on defining classes that represent objects with well-defined characteristics and functionality

- **Useful:** usable, easy to understand
- **Encapsulated**: show what is useful, hide what is hideable

Classes

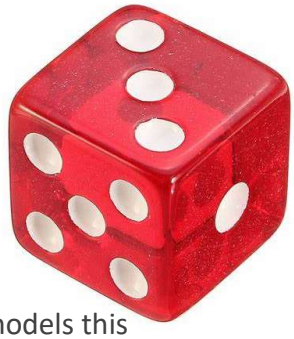
A class can contain data declarations and method definitions



Declare: say something exists, give it a name. Cannot use it until it is defined.
Define: say what something is equal to, give it a value.

Can have data definitions too!

Classes and Objects



An object has *state* and *behavior*

Consider a six-sided die (singular of dice)

- It's state can be defined as which face is showing
- It's primary behavior is that it can be rolled

We represent a die by designing a class called `Die` that models this state and behavior

- The class serves as the blueprint for a die object

We can then instantiate as many die objects as we need for any particular program

```
//********************************************************************
//  Die.java       Author: Lewis/Loftus
//
//  Represents one die (singular of dice) with faces showing values
//  between 1 and 6.
//********************************************************************

public class Die
{
   private final int MAX = 6;  // maximum face value

   private int faceValue;  // current value showing on the die

   //-----------------------------------------------------------------
   //  Constructor: Sets the initial face value.
   //-----------------------------------------------------------------
   public Die()
   {
      faceValue = 1;
   }
```

continue

First we will look at some code. Then we will explain it. Then we will look at another program.

CONSTRUCTOR: a method called automatically when an object is created.

PUBLIC and PRIVATE:

continue

```
   //-----------------------------------------------------------------
   //  Rolls the die and returns the result.
   //-----------------------------------------------------------------
   public int roll()
   {
      faceValue = (int)(Math.random() * MAX) + 1;
      return faceValue;
   }

   //-----------------------------------------------------------------
   //  Face value accessor.
   //-----------------------------------------------------------------
   public int getFaceValue()
   {
      return faceValue;
   }
```

continue

```
   //-----------------------------------------------------------------
   //  Returns a string representation of this die.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result = Integer.toString(faceValue);

      return result;
   }
}
```

```
//********************************************************************
//  RollingDice.java       Author: Lewis/Loftus
//
//  Demonstrates the creation and use of a user-defined class.
//********************************************************************

public class RollingDice
{
   //-----------------------------------------------------------------
   //  Creates two Die objects and rolls them several times.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      Die die1, die2;
      int sum;

      die1 = new Die();
      die2 = new Die();

      die1.roll();
      die2.roll();
      System.out.println("Die One: " + die1 + ", Die Two: " + die2);
    }
}
```

Classes

The values of the data define the state of an object created from the class

The functionality of the methods define the behaviors of the object

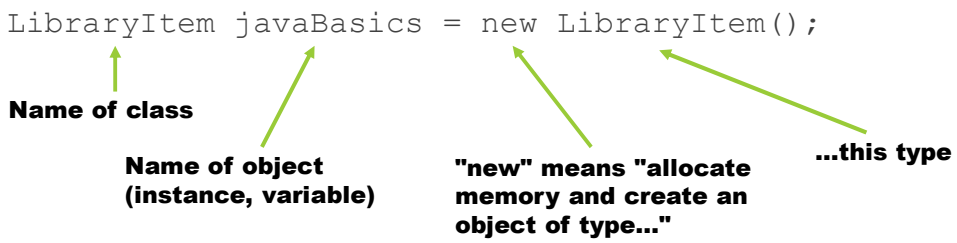
For our `Die` class, we declare an integer called `faceValue` that represents the current value showing on the face

One of the methods would "roll" the die by setting `faceValue` to a random number between one and six

- We **hide** the details of how rolling is implemented
- We **prevent** the user from changing the number of faces, current value, etc

Syntax: Creating Objects

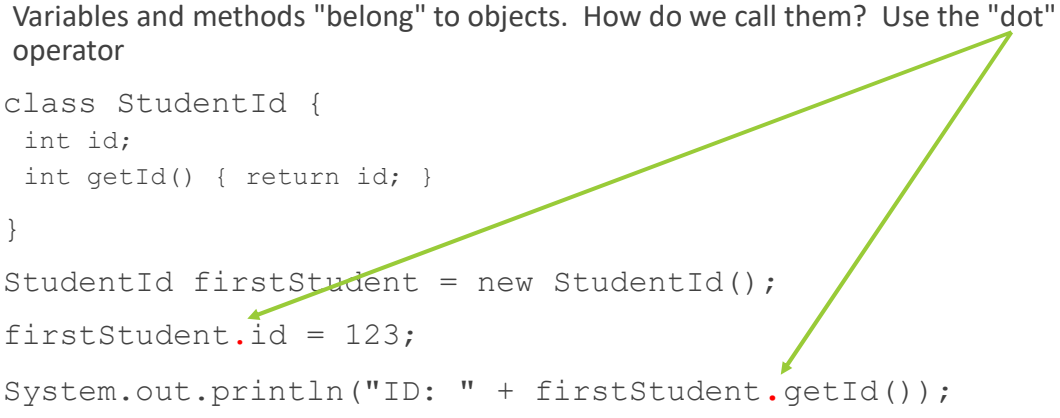
Particular Java syntax required. Assume `class LibraryItem` is defined:



And call Constructor!

Syntax: Accessing a Method or Variable

Variables and methods "belong" to objects. How do we call them? Use the "dot" operator



```
class StudentId {
 int id;
 int getId() { return id; }
}
StudentId firstStudent = new StudentId();
firstStudent.id = 123;
System.out.println("ID: " + firstStudent.getId());
```

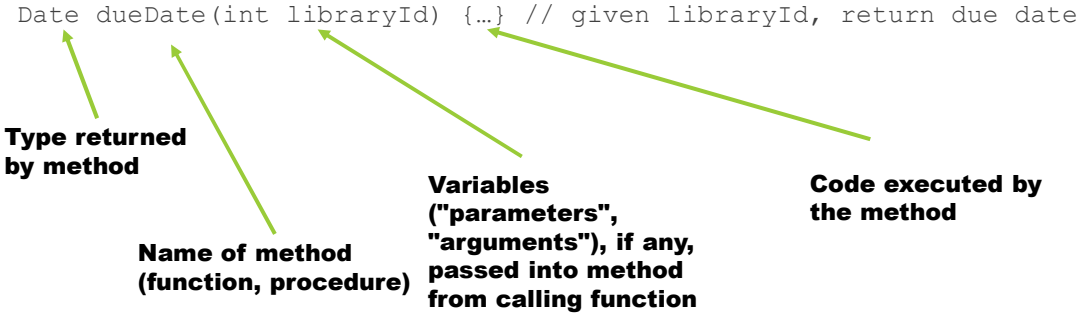
Why have getId() method?

Want to protect underlying variable, probably not make it available to be changed.

Probably have setId() method also: only allow to set 1x, make sure value is legal, etc.

Syntax: Defining Methods

```
Date dueDate(int libraryId) {…} // given libraryId, return due date
```



Return type is `void` if no value is returned by the method, e.g.

- `static public` **`void`** `main()`

Syntax: Defining Methods

A *method definition* specifies the code that will be executed when the method is invoked (called)

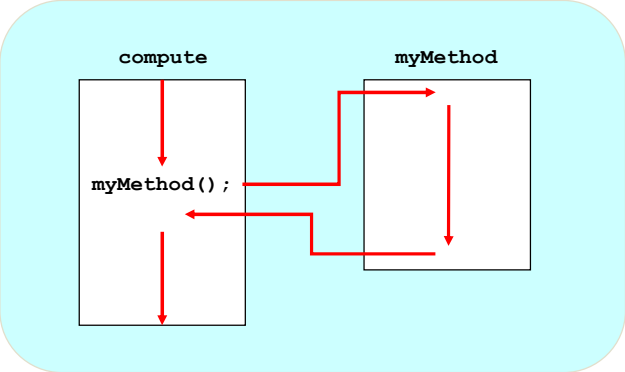
When a method is invoked, the flow of control jumps to the method and executes its code

When complete, the flow returns to the place where the method was called and continues

The invocation may or may not return a value, depending on how the method is defined

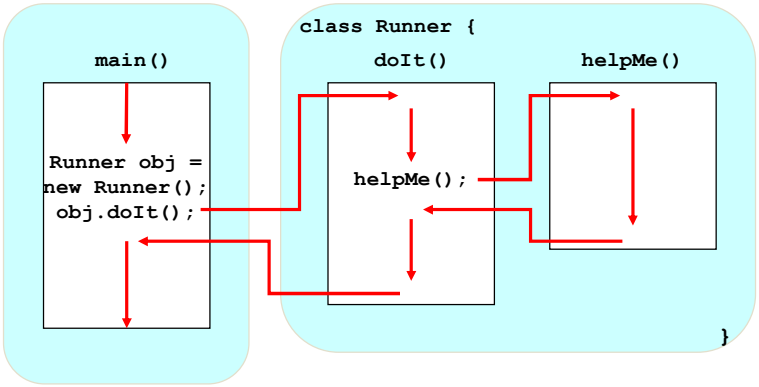
Method Flow of Execution

If the called method is in the same class, only the method name is needed



Method Flow of Execution

Outside of a class's own methods, a class's methods can be called via an object whose type is the class.



Method Parameters

When a method is called, the *actual parameters* from the caller are copied into the *formal parameters* in the method header

```
ch = obj.calc(2, count, "Hello");
```

```
char calc(int num1, int num2, String message)
{
   int sum = num1 + num2;
   char result = message.charAt(sum);

   return result;
}
```

Local Data

Local variables can be declared inside a method

The formal parameters of a method create *local variables* when the method is invoked

When the method finishes, all local variables are destroyed (including the formal parameters)

Instance variables, declared at the class level, exist as long as the object exists

The `return` Statement

The *return type* of a method indicates the type of value that the method sends back to the calling location

A method that does not return a value has a `void` return type

A *return statement* specifies the value that will be returned

```
return expression;
```

`expression` must conform to the return type

Instance Data

A variable declared at the class level (such as `faceValue`) is called *instance data*

Class methods can use:

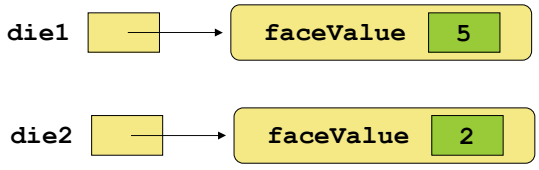
- Their local variables
- Their formal parameters (which become local variables)
- Instance variables

Each instance (object) has its own instance variables

- Each time a `Die` object is created, a new `faceValue` variable is created as well
- This is how two objects with same type can have different states

Instance Data

We can depict the two `Die` objects from the `RollingDice` program as follows:



Each object maintains its own `faceValue` variable, and thus its own state

The toString() Method

It's good practice to define a `toString()` method for a class

The `toString()` method returns a character string that represents the object in some way

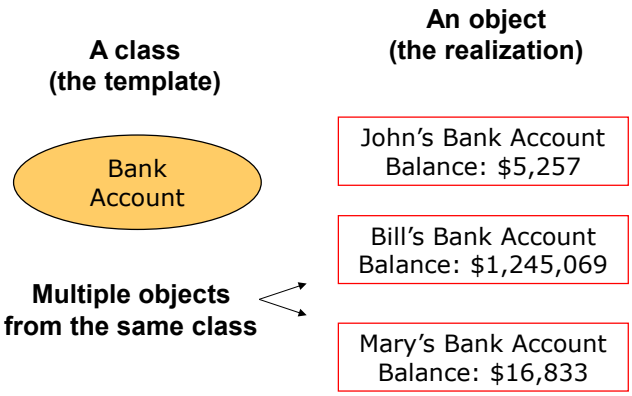
It is called automatically when an object is concatenated to a string or when it is passed to the `println()` method

It's also convenient for debugging problems

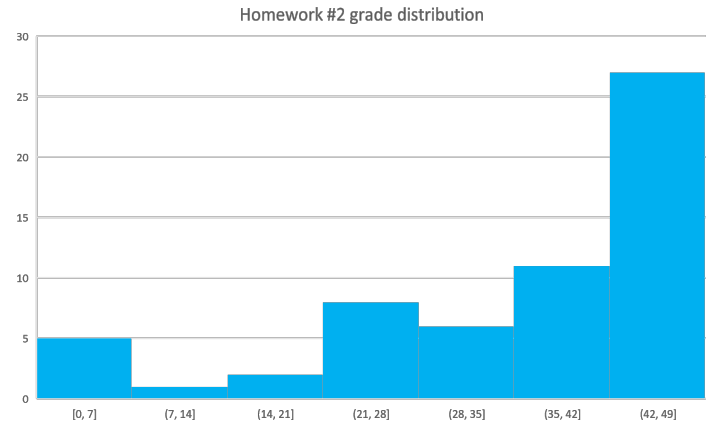
Let's look at more code…

BankAccount.java

Objects and Classes



Interlude



A huge spread in how much programming experience different people have coming into this course. Some have years and probably could teach the course. Some have very little. I heard from and saw a few people who really struggled with HW#3, not sure how to start. I would like to have a session for people who felt like they really struggled. We can review some concepts, go thru HW#3 together, and start on HW#4. If interested, please email me all times you are available this Thurs and Fri, and I will pick a time. Not sure if Zoom or in person.

Lab04

Let's take a look…